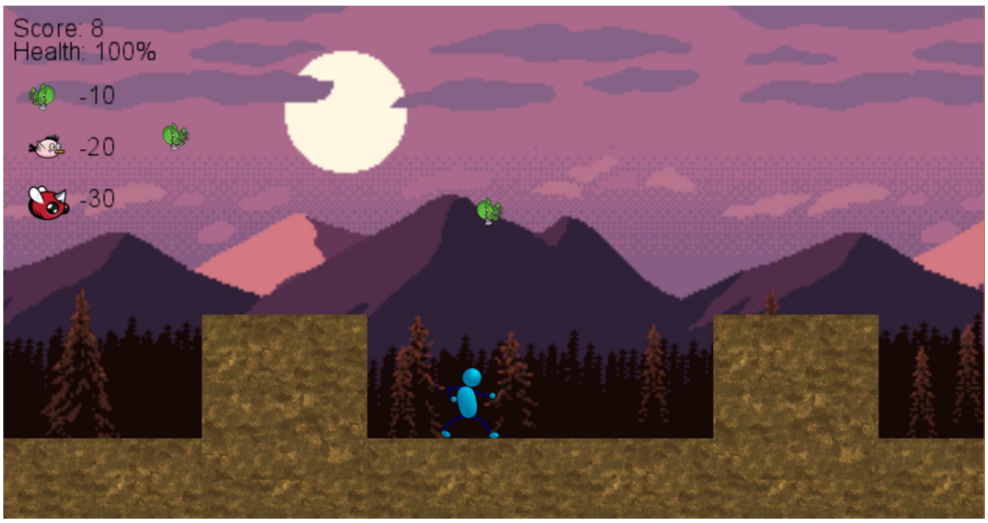
Rapport du Jeu GoblinDodger

Design du jeu :

Le jeu consiste à survivre les attaques des ennemis. Le joueur a une vie écrite sur l'écran débutant à 100%. Chaque ennemi lui enlève des vies selon leur type. Le joueur gagne quand il survie pour plus de 120 secondes.



Le code:

- 1 enum avec au moins 3 éléments

```
using Microsoft.Xna.Framework.Graphics;

namespace TestGame1.Obstacles.Moving
{
    enum EnemyType
    {
        Goblin, Bird, Fly
    }
```

```
switch (enemyLeft[enemyLeft.Count - 1].Level)
{
    case (int)EnemyType.Goblin:
        tempLeft.EnemyTexture = EnemyRight.content.Load<Texture2D>("Goblin/goblinLeft");
        tempLeft.EnemyRectangle = new Rectangle((int)tempLeft.EnemyPosition.X, (int)tempLeft.EnemyPosition.Y, tempLeft.EnemyTexture.Width, tempLeft.EnemyTextu
        break;
    case (int)EnemyType.Bird:
        tempLeft.EnemyTexture = EnemyRight.content.Load<Texture2D>("Bird/birdLeft");
        tempLeft.EnemyRectangle = new Rectangle((int)tempLeft.EnemyPosition.X, (int)tempLeft.EnemyPosition.Y, tempLeft.EnemyTexture.Width, tempLeft.EnemyTextu
        break;
    case (int)EnemyType.Fly:
        tempLeft.EnemyTexture = EnemyRight.content.Load<Texture2D>("Fly/flyLeft");
        tempLeft.EnemyRectangle = new Rectangle((int)tempLeft.EnemyPosition.X, (int)tempLeft.EnemyPosition.Y, tempLeft.EnemyTexture.Width, tempLeft.EnemyTextu
        break;
}
```

J’ai utilisé un enum incluant les noms de chaque type d’ennemi pour faciliter leur classification et utilisation dans le code. De même, ça aidait avec leur position dans les listes d’ennemis en les castant en int.

1 instruction d'itération (for, foreach, while)

```
if (state.IsKeyDown(Keys.Right) && canMoveRight)
{
    PlayerPosition.X += playerSpeed * gametime.ElapsedGameTime.Milliseconds;
    PlayerTexture = PlayerTextureRight;
}

if (state.IsKeyDown(Keys.Left) && canMoveLeft)
{
    PlayerPosition.X -= playerSpeed * gametime.ElapsedGameTime.Milliseconds;
    PlayerTexture = PlayerTextureLeft;
}
```

J'ai utilisé plusieurs instructions d'itérations durant le code principalement pour faire des actions pour chaque élément d'une liste que ce soit du background ou de la liste d'ennemis.

- 1 tableau

```
string[] stats = new string[4];
stats[0] = "You got hit by:";
stats[1] = "Goblins: 0";
stats[2] = "Bird: 0";
stats[3] = "Flies: 0";
```

J'ai utilisé un tableau pour garder en mémoire le texte du Scoreboard qui sera mis dans le fichier .txt, car cela faciliterait son impression sur le .txt.

- 1 appel de méthode avec 2 arguments nommés

```
public void Gravity(KeyboardState state, GameTime gametime)
{

    if ((state.IsKeyDown(Keys.Space) & !previousState.IsKeyDown(
        Keys.Space) || state.IsKeyDown(Keys.Up) & !previousState.IsKeyDown(
        Keys.Up)) & countJump < 2)
    {
        countJump++;
        isJumping = true;
        isFalling = false;
        timeJumping = 0;
    }
```

Pour but de faciliter la lecture de mon code, j'ai séparé des méthodes en plusieurs méthodes ce qui a causé plusieurs arguments dans certains dans elles. Par exemple, la méthode « Gravity » de la classe Player à besoin d'un KeyboardState et du Gametime.

- 1 définition de méthode avec 1 argument optionnel

```
public Enemy(Vector2 EnemyPosition, int Level = 1)
{
    this.Level = Level;
    this.EnemyPosition = EnemyPosition;
}
```

Pour faciliter mon testage du code, j'ai décidé de mettre la sorte d'ennemi comme optionnel pour pouvoir créer un Goblin par défaut et l'utiliser que pour mettre sa texture.

- 1 définition de méthode surchargée (overloaded)

```
public void PlayerRelativePosition(KeyboardState state, GameTime gametime, bool start) ...
public void PlayerRelativePosition(KeyboardState state, GameTime gametime, bool start, int end)...
```

J'ai créé deux méthodes appelées PlayerRelativePosition agissant dans deux différentes situations : avec la variable 'end' et sans cette dernière pour les différencier entre le début du plan et au milieu.

- 1 héritage de classes :
    - 1 classe abstraite de base
      J'ai dû créer une classe abstraite pour le type de fond d'écran pour faciliter leur implémentation.
    - 3 classes dérivées
      Comme la classe abstraite du fond d'écran a été créé, j'ai créé 6 classes dérivées.
    - 1 méthode à substituer (virtual, override)
      Les méthodes Draw, Load et Update sont virtual dans la classe abstraite et override dans les classes dérivées.
    - 1 utilisation de polymorphisme
      La classe Enemy contient des variables qui sont utilisées par les classes dérivés lors de la création de leur instance comme EnemyPosition.

```
abstract class BackgroundTiles
{
```

```
class BG : BackgroundTiles
{
    public new static Texture2D BackgroundLayer;
    public static float tilespeed = 0.010f;
    public BG(string path, Vector2 position) : base(path, position)
    {
    }
```

- 1 délégué et 1 événement

```
public delegate void MovementHandler(GameTime gametime);

public MovementHandler PlayerMoves;
```

J'ai fait l'évènement que lorsque il y a movement(si le joueur est sautant ou tombant) leur respective méthode s'exécute.

- 1 collection utilisée (List, HashSet, Dictionary)

```
public static List<List<BackgroundTiles>> backgroundTiles = new List<List<BackgroundTiles>>();

public void Initialize()
{

    backgroundTiles.Add(new List<BackgroundTiles>());
    backgroundTiles.Add(new List<BackgroundTiles>());
    backgroundTiles.Add(new List<BackgroundTiles>());
    backgroundTiles.Add(new List<BackgroundTiles>());
    backgroundTiles.Add(new List<BackgroundTiles>());

}
```

J'ai utilisés des Lists pour mieux classer les fonds d'écrans et ennemis.

- 1 donnée sauvegardée

```
StreamReader docRead;
FileStream fileAppend;

string[] stats = new string[4];
static public string filename = @"Stats\Stats.txt";
public int timeTospawn;
public int switch1;
public Random random = new Random();
public int maxLevel = 1;
public override void Load(ContentManager content)
{

    filename = Path.GetFullPath(filename);
    fileAppend = new FileStream(filename, FileMode.Create);

    Directory.CreateDirectory(Path.GetDirectoryName(filename));
    doc = new StreamWriter(fileAppend);
```

J'ai utilisé un .txt pour sauvegarder les statistiques d'un joueur de façon à ce que le testeur de collisions écrive dessus et que le scoreboard le lise.

Le jeu

- Doit utiliser au moins le clavier ou la souris pour les contrôles
On peut bouger avec les flèches et sauter avec space ou la flèche en haut.

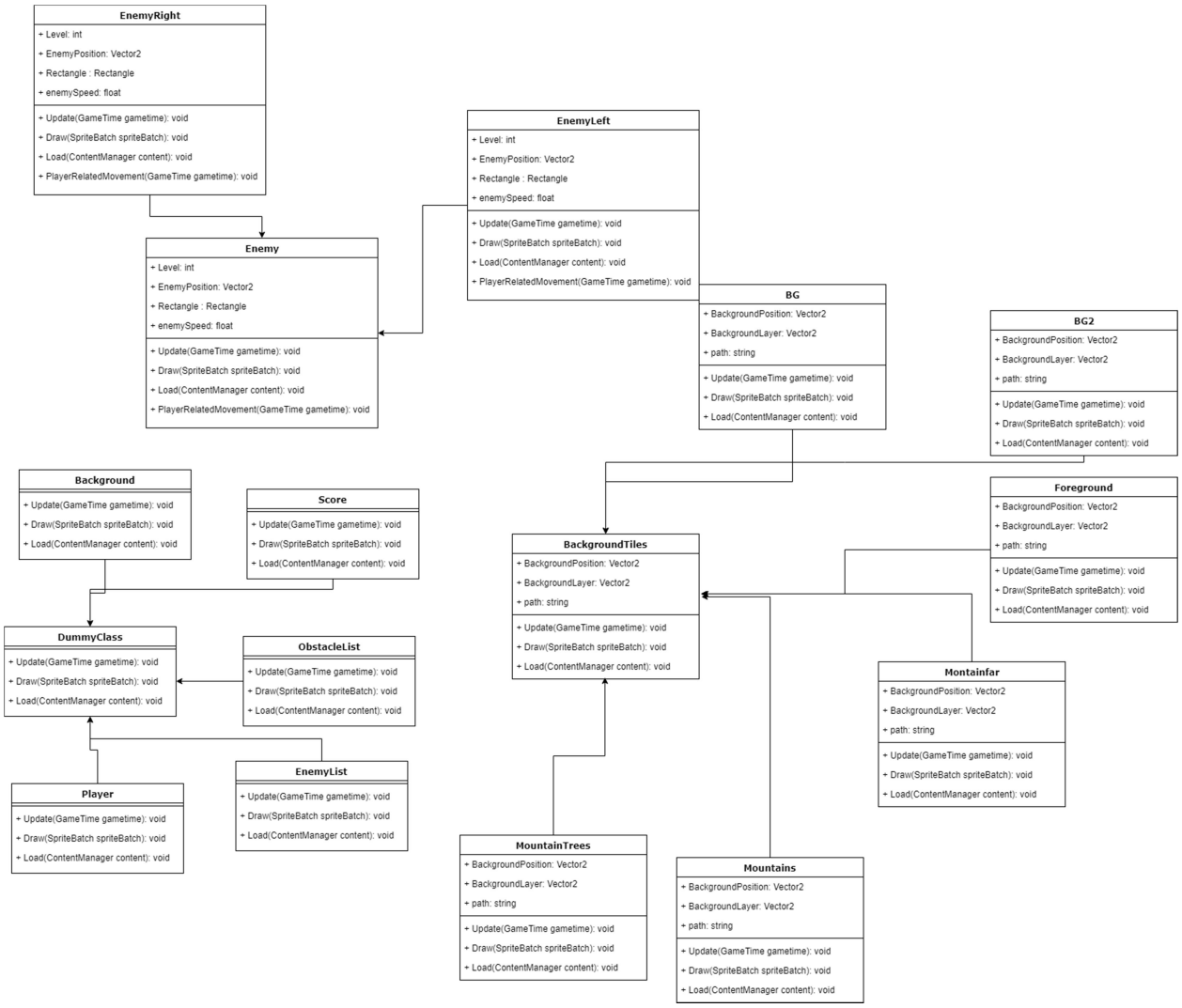
- 1 minute de jeu (gameplay)
Si vous êtes assez bon, il doit y en avoir. :wink:

- 3 textures différentes
Il y a une texture pour le personnage, 3 ennemis, 6 fonds d'écran et le sol.

- 1 détection de collision
Il y a une détection de collisions entre un Ennemi et le Joueur et ce dernier avec les obstacles qui ne bougent pas (sol et obstacles).

- Utilisation de la classe GameTime pour le temps
Le mouvement dans toutes les classes utilise le GameTime pour être plus précis et pour calculer la vitesse avec l'accélération.

Hiérarchie des classes :



La classe la plus basique est le DummyClass qui inclut les méthodes d'Update, Draw et Load. J'ai utilisé les mêmes méthodes comme base pour Enemy et BackgroundTiles, mais j'ai rajouté quelques variables uniques pour les classes dérivées qui inclurent la position et la texture.

Difficultés rencontrées :

○ Au moins une discussion sur le développement d'un algorithme pour résoudre un problème en particulier.

J'ai eu de la difficulté avec le mouvement et les collisions à cause de la fréquence des Updates et Draws. Pour combler le fait que la fréquence était lente, j'ai fait que le personnage avance d'un certain nombre de positions à la fois. Cela causait des problèmes si l'obstacle était à une position dont le personnage ne pouvait jamais toucher. Donc j'ai dû changer la logique d'une position exacte à un intervalle pour que le personnage puisse faire l'illusion de toucher l'obstacle.

○ Au moins une discussion sur une difficulté liée à la langue de programmation

J'avais un problème lors de la création des nombres d'ennemis. À chaque certain nombre de secondes, un ennemi apparaît dans l'écran. Le problème a été qu'en créant un ennemi dans l'update, on ne pouvait pas load une nouvelle texture tout de suite sans changer les autres. Ce qui a causé que je crée deux différentes classes : EnemyLeft, EnemyRight pour les deux sortes de textures pour chaque côté et l'enum EnemyType pour différencier les types d'ennemis.

Sources:

Player:
https://openclipart.org/detail/14662/stickman

Background:
https://opengameart.org/content/mountain-at-dusk-background

Grass:
https://opengameart.org/content/grass-textures-tiles

Bird:
https://opengameart.org/content/bevouliin-free-flappy-pink-bird-spritesheets-for-game-developers

Goblin:
https://opengameart.org/content/2d-goblin-chibi

Fly:
https://opengameart.org/content/red-horned-bee-fly-game-sprites-character

ScoreBoard:
https://opengameart.org/content/wooden-box-2d